ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ №4

Выполнил:
Батуров Максим Дмитриевич
Группа: 6203-010302D

Оглавление

Задание №1

В рамках первого задания я реализовал конструкторы для классов ArrayTabulatedFunction и LinkedListTabulatedFunction, которые принимают массив точек FunctionPoint. Конструкторы выполняют строгие проверки на минимальное количество точек и на упорядоченность абсцисс. Для сравнения вещественных чисел использовался машинный эпсилон. При нарушении условий генерируется исключение IllegalArgumentException.

```
// конструктор, получающий массив точек
public ArrayTabulatedFunction(FunctionPoint[] points) {  2 usages
    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    // проверка упорядоченности точек по X
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i].getX() >= points[i + 1].getX() - EPSILON) {
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
        }
    }

    this.pointsCount = points.length;
    this.points = new FunctionPoint[pointsCount + 3];

    // создаем копии точек для обеспечения инкапсуляции
    for (int i = 0; i < pointsCount; i++) {
        this.points[i] = new FunctionPoint(points[i]);
    }
}
```

```java
public LinkedListTabulatedFunction(FunctionPoint[] points) {  no usages
    this(); // вызов конструктора по умолчанию для инициализации головы

    if (points.length < 2) {
        throw new IllegalArgumentException("Количество точек должно быть не менее 2");
    }

    // проверка упорядоченности точек по X
    for (int i = 0; i < points.length - 1; i++) {
        if (points[i].getX() >= points[i + 1].getX() - EPSILON) { // используем EPSILON
            throw new IllegalArgumentException("Точки должны быть упорядочены по возрастанию X");
        }
    }

    // добавляем точки в список, создавая копии для инкапсуляции
    for (FunctionPoint point : points) {
        addNodeToTail(new FunctionPoint(point));
    }
}
```

Задание №2

Был разработан интерфейс Function, определяющий основные операции для функций одной переменной: получение левой и правой границ области определения и вычисление значения в заданной точке. Интерфейс TabulatedFunction был переработан и теперь наследует от Function.

```java
package functions;

public interface Function {
    // возвращает значение левой границы обл. опр. ф-ции
    double getLeftDomainBorder();  11 implementations

    // возвращает значение правой границы обл. опр. ф-ции
    double getRightDomainBorder();  11 implementations

    // возвращает значение ф-ции в заданной точке
    double getFunctionValue(double x);  13 implementations
}
```

Задание №3

В пакете functions.basic созданы классы для аналитически заданных функций. Класс Exp реализует экспоненциальную функцию с использованием Math.exp(). Класс Log вычисляет логарифм по заданному основанию через натуральный логарифм Math.log(). Для тригонометрических функций разработан абстрактный базовый класс TrigonometricFunction с общей областью определения, от которого наследуются классы Sin, Cos и Tan.

```java
package functions.basic;

import functions.Function;

public class Exp implements Function {  2 usages

    // возвращает значение экспоненты в точке x
    public double getFunctionValue(double x) {
        return Math.exp(x);
    }

    // возвращает левую границу
    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    // возвращает правую границу
    public double getRightDomainBorder() {
        return Double.POSITIVE_INFINITY;
    }
}
```

```java
package functions.basic;

import functions.Function;

// класс для триг ф-ций
public abstract class TrigonometricFunction implements Function {  3 usages  3 inhe

    public double getLeftDomainBorder() {
        return Double.NEGATIVE_INFINITY;
    }

    public double getRightDomainBorder() { return Double.POSITIVE_INFINITY; }
}
```

Задание №4

В пакете functions.meta реализованы классы для комбинирования функций. Класс Sum вычисляет сумму двух функций с областью определения как пересечение исходных областей. Класс Mult реализует произведение функций.

Класс Power выполняет возведение функции в заданную степень. Класс Scale обеспечивает масштабирование по осям координат. Класс Shift реализует сдвиг функции вдоль осей. Класс Composition создает композицию двух функций.

Задание №5

Создан служебный класс Functions с приватным конструктором, предотвращающим создание экземпляров. Класс содержит статические методы shift, scale, power, sum, mult и composition, которые возвращают объекты соответствующих функций из пакета functions.meta. Это обеспечивает удобный и безопасный способ создания сложных функций.

```java
package functions;

import functions.meta.*;

public class Functions {  7 usages

    // коснтруктор чтобы нельзя было создать объект класса
    private Functions() {}  no usages

    public static Function shift(Function f, double shiftX, double shiftY) { return new Shift(f, shiftX, shiftY); }

    public static Function scale(Function f, double scaleX, double scaleY) { return new Scale(f, scaleX, scaleY); }

    public static Function power(Function f, double power) { return new Power(f, power); }

    public static Function sum(Function f1, Function f2) { return new Sum(f1, f2); }

    public static Function mult(Function f1, Function f2) { return new Mult(f1, f2); }

    public static Function composition(Function f1, Function f2) { return new Composition(f1, f2); }
}
```

Задание №6

Разработан класс TabulatedFunctions с приватным конструктором, содержащий метод tabulate для создания табулированных аналогов аналитических функций. При нарушении условий генерируется IllegalArgumentException. Метод возвращает ArrayTabulatedFunction с равномерно распределенными точками.

```java
package functions;

import java.io.*;

public class TabulatedFunctions {  11 usages
    private TabulatedFunctions() {}  no usages

    public static TabulatedFunction tabulate(Function function, double leftX, double rightX, int pointsCount) {
        // проверка границ
        if (leftX < function.getLeftDomainBorder() || rightX > function.getRightDomainBorder()) {
            throw new IllegalArgumentException("Границы табулирования выходят за область определения функции");
        }

        // проверка минимального количества точек
        if (pointsCount < 2) {
            throw new IllegalArgumentException("Количество точек должно быть не менее 2");
        }

        // проверка интервала
        if (leftX >= rightX) {
            throw new IllegalArgumentException("Левая граница должна быть меньше правой");
        }

        // создаем массив значений ф-ции
        double[] values = new double[pointsCount];
        double step = (rightX - leftX) / (pointsCount - 1);

        // вычисляем значения ф-ции
        for (int i = 0; i < pointsCount; i++) {
            double x = leftX + i * step;
            values[i] = function.getFunctionValue(x);
        }

        // возвращаем табулированную ф-цию
        return new ArrayTabulatedFunction(leftX, rightX, values);
    }
```

Задание №7

Реализованы методы для работы с потоками ввода-вывода. Для байтовых потоков используются DataOutputStream и DataInputStream, обеспечивающие эффективную запись примитивных типов. Для текстовых потоков применяются PrintWriter и StreamTokenizer с пробелами в качестве разделителей. Исключения IOException выбрасываются вызывающему коду для гибкой обработки ошибок, при этом потоки не закрываются внутри методов для возможности дальнейшего использования.

```java
// вывод табулированной ф-ции в байтовый поток
public static void outputTabulatedFunction(TabulatedFunction function, OutputStream out) throws IOException {
    DataOutputStream dataOut = new DataOutputStream(out);

    // записываем кол-во точек
    dataOut.writeInt(function.getPointsCount());

    // записываем координаты всех точек
    for (int i = 0; i < function.getPointsCount(); i++) {
        dataOut.writeDouble(function.getPointX(i));
        dataOut.writeDouble(function.getPointY(i));
    }

    dataOut.flush();
}

// ввод табулированной функции из байтового потока
public static TabulatedFunction inputTabulatedFunction(InputStream in) throws IOException {  1 usage
    DataInputStream dataIn = new DataInputStream(in);

    // читаем кол-во точек
    int pointsCount = dataIn.readInt();

    // читаем координаты точек
    FunctionPoint[] points = new FunctionPoint[pointsCount];
    for (int i = 0; i < pointsCount; i++) {
        double x = dataIn.readDouble();
        double y = dataIn.readDouble();
        points[i] = new FunctionPoint(x, y);
    }

    // создаем табулированную ф-цию
    return new ArrayTabulatedFunction(points);
}
```

```java
// запись ф-ции в символьный поток
public static void writeTabulatedFunction(TabulatedFunction function, Writer out) throws IOException {
    PrintWriter writer = new PrintWriter(out);

    // записываем кол-во точек
    writer.print(function.getPointsCount());
    writer.print(" ");

    for (int i = 0; i < function.getPointsCount(); i++) {
        writer.print(function.getPointX(i));
        writer.print(" ");
        writer.print(function.getPointY(i));
        if (i < function.getPointsCount() - 1) {
            writer.print(" ");
        }
    }

    writer.flush();
}
```

```java
    // считывание ф-ции из символьного потока
    public static TabulatedFunction readTabulatedFunction(Reader in) throws IOException {

        if (in == null) {
            throw new IllegalArgumentException("передан нулевой поток ввода");
        }

        StreamTokenizer tokenizer = new StreamTokenizer(in);
        tokenizer.parseNumbers();

        // считываем число точек
        int tokenType = tokenizer.nextToken();
        if (tokenType != StreamTokenizer.TT_NUMBER) {
            throw new IOException("не найдено количество точек");
        }
        int count = (int) tokenizer.nval;

        // создаем массив для хранения точек
        FunctionPoint[] functionPoints = new FunctionPoint[count];

        // считываем пары
        for (int i = 0; i < count; i++) {
            // считываем x координату
            tokenType = tokenizer.nextToken();
            if (tokenType != StreamTokenizer.TT_NUMBER) {
                throw new IOException("отсутствует x координата для точки " + i);
            }
            double xCoord = tokenizer.nval;

            // считываем y координату
            tokenType = tokenizer.nextToken();
            if (tokenType != StreamTokenizer.TT_NUMBER) {
                throw new IOException("отсутствует y координата для точки " + i);
            }
            double yCoord = tokenizer.nval;

            // создаем точку с полученными координатами
            functionPoints[i] = new FunctionPoint(xCoord, yCoord);
        }

        // возвращаем новую табулированную ф-цию
        return new ArrayTabulatedFunction(functionPoints);
    }
}
```

Задание №8

В классе Main проведено комплексное тестирование функциональности. Протестированы базовые функции Sin и Cos, созданы их табулированные аналоги и проведено сравнение точности. Исследована сумма квадратов табулированных синуса и косинуса. Проверена работа с файлами: экспонента сохранена и загружена из текстового файла, логарифм - из бинарного. Также

понял, что бинарный формат более эффективен по объему, а текстовый более читаем.

```
1. Sin и Cos на отрезке [0, pi] с шагом 0.1:
Sin(x):
sin(0,0) = 0,000000
sin(0,1) = 0,099833
sin(0,2) = 0,198669
sin(0,3) = 0,295520
sin(0,4) = 0,389418
sin(0,5) = 0,479426
sin(0,6) = 0,564642
sin(0,7) = 0,644218
sin(0,8) = 0,717356
sin(0,9) = 0,783327
sin(1,0) = 0,841471
sin(1,1) = 0,891207
sin(1,2) = 0,932039
sin(1,3) = 0,963558
sin(1,4) = 0,985450
sin(1,5) = 0,997495
sin(1,6) = 0,999574
sin(1,7) = 0,991665
sin(1,8) = 0,973848
sin(1,9) = 0,946300
sin(2,0) = 0,909297
sin(2,1) = 0,863209
sin(2,2) = 0,808496
sin(2,3) = 0,745705
sin(2,4) = 0,675463
sin(2,5) = 0,598472
sin(2,6) = 0,515501
sin(2,7) = 0,427380
sin(2,8) = 0,334988
sin(2,9) = 0,239249
sin(3,0) = 0,141120
sin(3,1) = 0,041581
```

```
Cos(x):
cos(0,0) = 1,000000
cos(0,1) = 0,995004
cos(0,2) = 0,980067
cos(0,3) = 0,955336
cos(0,4) = 0,921061
cos(0,5) = 0,877583
cos(0,6) = 0,825336
cos(0,7) = 0,764842
cos(0,8) = 0,696707
cos(0,9) = 0,621610
cos(1,0) = 0,540302
cos(1,1) = 0,453596
cos(1,2) = 0,362358
cos(1,3) = 0,267499
cos(1,4) = 0,169967
cos(1,5) = 0,070737
cos(1,6) = -0,029200
cos(1,7) = -0,128844
cos(1,8) = -0,227202
cos(1,9) = -0,323290
cos(2,0) = -0,416147
cos(2,1) = -0,504846
cos(2,2) = -0,588501
cos(2,3) = -0,666276
cos(2,4) = -0,737394
cos(2,5) = -0,801144
cos(2,6) = -0,856889
cos(2,7) = -0,904072
cos(2,8) = -0,942222
cos(2,9) = -0,970958
cos(3,0) = -0,989992
cos(3,1) = -0,999135
```

```
2. Табулированные аналоги Sin и Cos (10 точек):
Сравнение точных и табулированных значений:
x=0,0: sin=0,000000 (tab=0,000000, err=0,000000) | cos=1,000000 (tab=1,000000, err=0,000000)
x=0,1: sin=0,099833 (tab=0,097982, err=0,001852) | cos=0,995004 (tab=0,982723, err=0,012281)
x=0,2: sin=0,198669 (tab=0,195963, err=0,002706) | cos=0,980067 (tab=0,965446, err=0,014620)
x=0,3: sin=0,295520 (tab=0,293945, err=0,001576) | cos=0,955336 (tab=0,948170, err=0,007167)
x=0,4: sin=0,389418 (tab=0,385907, err=0,003512) | cos=0,921061 (tab=0,914355, err=0,006706)
x=0,5: sin=0,479426 (tab=0,472070, err=0,007355) | cos=0,877583 (tab=0,864608, err=0,012974)
x=0,6: sin=0,564642 (tab=0,558234, err=0,006409) | cos=0,825336 (tab=0,814862, err=0,010474)
x=0,7: sin=0,644218 (tab=0,643982, err=0,000235) | cos=0,764842 (tab=0,764620, err=0,000222)
x=0,8: sin=0,717356 (tab=0,707935, err=0,009421) | cos=0,696707 (tab=0,688404, err=0,008302)
x=0,9: sin=0,783327 (tab=0,771888, err=0,011439) | cos=0,621610 (tab=0,612188, err=0,009422)
x=1,0: sin=0,841471 (tab=0,835841, err=0,005630) | cos=0,540302 (tab=0,535972, err=0,004330)
x=1,1: sin=0,891207 (tab=0,883993, err=0,007214) | cos=0,453596 (tab=0,450633, err=0,002963)
x=1,2: sin=0,932039 (tab=0,918022, err=0,014017) | cos=0,362358 (tab=0,357141, err=0,005217)
x=1,3: sin=0,963558 (tab=0,952051, err=0,011508) | cos=0,267499 (tab=0,263648, err=0,003851)
x=1,4: sin=0,985450 (tab=0,984808, err=0,000642) | cos=0,169967 (tab=0,169931, err=0,000037)
x=1,5: sin=0,997495 (tab=0,984808, err=0,012687) | cos=0,070737 (tab=0,070437, err=0,000300)
x=1,6: sin=0,999574 (tab=0,984808, err=0,014766) | cos=-0,029200 (tab=-0,029056, err=0,000144)
x=1,7: sin=0,991665 (tab=0,984808, err=0,006857) | cos=-0,128844 (tab=-0,128549, err=0,000296)
x=1,8: sin=0,973848 (tab=0,966204, err=0,007644) | cos=-0,227202 (tab=-0,224761, err=0,002441)
x=1,9: sin=0,946300 (tab=0,932175, err=0,014125) | cos=-0,323290 (tab=-0,318254, err=0,005035)
x=2,0: sin=0,909297 (tab=0,898147, err=0,011151) | cos=-0,416147 (tab=-0,411747, err=0,004400)
x=2,1: sin=0,863209 (tab=0,862441, err=0,000768) | cos=-0,504846 (tab=-0,504272, err=0,000574)
x=2,2: sin=0,808496 (tab=0,798488, err=0,010008) | cos=-0,588501 (tab=-0,580488, err=0,008013)
x=2,3: sin=0,745705 (tab=0,734535, err=0,011170) | cos=-0,666276 (tab=-0,656704, err=0,009572)
x=2,4: sin=0,675463 (tab=0,670582, err=0,004881) | cos=-0,737394 (tab=-0,732920, err=0,004474)
x=2,5: sin=0,598472 (tab=0,594072, err=0,004401) | cos=-0,801144 (tab=-0,794171, err=0,006973)
x=2,6: sin=0,515501 (tab=0,507908, err=0,007593) | cos=-0,856889 (tab=-0,843917, err=0,012972)
x=2,7: sin=0,427380 (tab=0,421745, err=0,005635) | cos=-0,904072 (tab=-0,893664, err=0,010408)
x=2,8: sin=0,334988 (tab=0,334698, err=0,000290) | cos=-0,942222 (tab=-0,940984, err=0,001239)
x=2,9: sin=0,239249 (tab=0,236716, err=0,002533) | cos=-0,970958 (tab=-0,958261, err=0,012698)
x=3,0: sin=0,141120 (tab=0,138735, err=0,002385) | cos=-0,989992 (tab=-0,975537, err=0,014455)
x=3,1: sin=0,041581 (tab=0,040753, err=0,000828) | cos=-0,999135 (tab=-0,992814, err=0,006321)
```

```
Точки табулированного Sin:
(0,0000, 0,000000) (0,3491, 0,342020) (0,6981, 0,642788) (1,0472, 0,866025) (1,3963, 0,984808) (1,7453, 0,984808) (2,0944, 0,866025) (2,4435, 0,642788) (2,7925, 0,342020) (3,1416, 0,000000)

Точки табулированного Cos:
(0,0000, 1,000000) (0,3491, 0,939693) (0,6981, 0,766044) (1,0472, 0,500000) (1,3963, 0,173648) (1,7453, -0,173648) (2,0944, -0,500000) (2,4435, -0,766044) (2,7925, -0,939693) (3,1416, -1,000000)
```

```
3. Сумма квадратов табулированных Sin и Cos:
sin^2(x) + cos^2(x) (должно быть близко к 1):
x=0,0: sin^2 + cos^2 = 1,00000000
x=0,1: sin^2 + cos^2 = 0,97534529
x=0,2: sin^2 + cos^2 = 0,97048832
x=0,3: sin^2 + cos^2 = 0,98542910
x=0,4: sin^2 + cos^2 = 0,98496848
x=0,5: sin^2 + cos^2 = 0,97039758
x=0,6: sin^2 + cos^2 = 0,97562443
x=0,7: sin^2 + cos^2 = 0,99935789
x=0,8: sin^2 + cos^2 = 0,97507306
x=0,9: sin^2 + cos^2 = 0,97058598
x=1,0: sin^2 + cos^2 = 0,98589664
x=1,1: sin^2 + cos^2 = 0,98451477
x=1,2: sin^2 + cos^2 = 0,97031375
x=1,3: sin^2 + cos^2 = 0,97591048
x=1,4: sin^2 + cos^2 = 0,99872269
x=1,5: sin^2 + cos^2 = 0,97480774
x=1,6: sin^2 + cos^2 = 0,97069054
x=1,7: sin^2 + cos^2 = 0,98637108
x=1,8: sin^2 + cos^2 = 0,98406796
x=1,9: sin^2 + cos^2 = 0,97023683
x=2,0: sin^2 + cos^2 = 0,97620344
x=2,1: sin^2 + cos^2 = 0,99809440
x=2,2: sin^2 + cos^2 = 0,97454934
x=2,3: sin^2 + cos^2 = 0,97080201
x=2,4: sin^2 + cos^2 = 0,98685244
x=2,5: sin^2 + cos^2 = 0,98362807
x=2,6: sin^2 + cos^2 = 0,97016682
x=2,7: sin^2 + cos^2 = 0,97650331
x=2,8: sin^2 + cos^2 = 0,99747303
x=2,9: sin^2 + cos^2 = 0,97429784
x=3,0: sin^2 + cos^2 = 0,97092040
x=3,1: sin^2 + cos^2 = 0,98734070

Исследование влияния количества точек на точность:
Точек: 10, максимальное отклонение от 1: 0,02983318
Точек: 20, максимальное отклонение от 1: 0,00681713
Точек: 30, максимальное отклонение от 1: 0,00291407
```

```
4. Экспонента и текстовый файл:
Сравнение исходной и прочитанной экспоненты:
x=0,0: исходная=1,000000, из файла=1,000000, совпадают=true
x=1,0: исходная=2,718282, из файла=2,718282, совпадают=true
x=2,0: исходная=7,389056, из файла=7,389056, совпадают=true
x=3,0: исходная=20,085537, из файла=20,085537, совпадают=true
x=4,0: исходная=54,598150, из файла=54,598150, совпадают=true
x=5,0: исходная=148,413159, из файла=148,413159, совпадают=true
x=6,0: исходная=403,428793, из файла=403,428793, совпадают=true
x=7,0: исходная=1096,633158, из файла=1096,633158, совпадают=true
x=8,0: исходная=2980,957987, из файла=2980,957987, совпадают=true
x=9,0: исходная=8103,083928, из файла=8103,083928, совпадают=true
x=10,0: исходная=22026,465795, из файла=22026,465795, совпадают=true

5. Логарифм и текстовый файл:
Сравнение исходного и прочитанного логарифма:
x=0,1: исходный=-2,302585, из файла=-2,302585, совпадают=true
x=1,1: исходный=0,086178, из файла=0,086178, совпадают=true
x=2,1: исходный=0,732368, из файла=0,732368, совпадают=true
x=3,1: исходный=1,121678, из файла=1,121678, совпадают=true
x=4,1: исходный=1,401183, из файла=1,401183, совпадают=true
x=5,1: исходный=1,619388, из файла=1,619388, совпадают=true
x=6,0: исходный=1,798404, из файла=1,798404, совпадают=true
x=7,0: исходный=1,950187, из файла=1,950187, совпадают=true
x=8,0: исходный=2,081938, из файла=2,081938, совпадают=true
x=9,0: исходный=2,198335, из файла=2,198335, совпадают=true
x=10,0: исходный=2,302585, из файла=2,302585, совпадают=true
```

Задание №9

Реализована сериализация табулированных функций двумя способами. LinkedListTabulatedFunction использует стандартный механизм Serializable, что удобно, но может приводить к избыточности данных. ArrayTabulatedFunction реализует Externalizable, что дает полный контроль над процессом сериализации. Внешняя сериализация позволяет сохранять только существенные данные - количество точек и их координаты, игнорируя служебные поля и структуры. Это обеспечивает компактность и эффективность, а также независимость от внутренней реализации классов.

```
public class ArrayTabulatedFunction implements TabulatedFunction, Externalizable {  7 us
    private FunctionPoint[] points;  37 usages
    private int pointsCount;  37 usages
    private static final double EPSILON = 1e-10;  12 usages
```

```java
public void writeExternal(ObjectOutput out) throws IOException {
    out.writeInt(pointsCount);
    for (int i = 0; i < pointsCount; i++) {
        out.writeDouble(points[i].getX());
        out.writeDouble(points[i].getY());
    }
}

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    int count = in.readInt();
    points = new FunctionPoint[count + 3];
    pointsCount = count;
    for (int i = 0; i < count; i++) {
        double x = in.readDouble();
        double y = in.readDouble();
        points[i] = new FunctionPoint(x, y);
    }
}
```

```java
public class LinkedListTabulatedFunction implements TabulatedFunction, Serializable {
    private FunctionNode head;  20 usages
    private int pointcount;  22 usages
    private static final double EPSILON = 1e-10;  12 usages

    private static class FunctionNode implements Serializable {  25 usages
        private FunctionPoint point;  3 usages
        private FunctionNode prev;  3 usages
        private FunctionNode next;  3 usages

        public FunctionNode(FunctionPoint point) {  3 usages
            this.point = point; // устанавливаем точку
            this.prev = null; // предыдущий узел пока null
            this.next = null; // следующий узел пока null
        }

        public FunctionPoint getPoint() {  17 usages
            return point; // возвращаем точку
        }

        public void setPoint(FunctionPoint point) {  2 usages
            this.point = point; // устанавливаем новую точку
        }

        public FunctionNode getPrev() {  6 usages
            return prev; // возвращаем предыдущий узел
        }

        public void setPrev(FunctionNode prev) {  6 usages
            this.prev = prev; // устанавливаем предыдущий узел
        }

        public FunctionNode getNext() {  13 usages
            return next; // возвращаем следующий узел
        }

        public void setNext(FunctionNode next) {  6 usages
            this.next = next; // устанавливаем следующий узел
        }
    }
```

```
Сериализация
Функция сериализована в serializable.ser
Функция десериализована из serializable.ser
Сравнение после Serializable:
x=0,0: исходная=NaN, восстановленная=NaN, совпадают=false
x=1,0: исходная=1,000000, восстановленная=1,000000, совпадают=true
x=2,0: исходная=2,000000, восстановленная=2,000000, совпадают=true
x=3,0: исходная=3,000000, восстановленная=3,000000, совпадают=true
x=4,0: исходная=4,000000, восстановленная=4,000000, совпадают=true
x=5,0: исходная=5,000000, восстановленная=5,000000, совпадают=true
x=6,0: исходная=6,000000, восстановленная=6,000000, совпадают=true
x=7,0: исходная=7,000000, восстановленная=7,000000, совпадают=true
x=8,0: исходная=8,000000, восстановленная=8,000000, совпадают=true
x=9,0: исходная=9,000000, восстановленная=9,000000, совпадают=true
x=10,0: исходная=10,000000, восстановленная=10,000000, совпадают=true

Экстернализация
Функция сериализована в externalizable.ser
Функция десериализована из externalizable.ser
Сравнение после Externalizable:
x=0,0: исходная=NaN, восстановленная=NaN, совпадают=false
x=1,0: исходная=1,000000, восстановленная=1,000000, совпадают=true
x=2,0: исходная=2,000000, восстановленная=2,000000, совпадают=true
x=3,0: исходная=3,000000, восстановленная=3,000000, совпадают=true
x=4,0: исходная=4,000000, восстановленная=4,000000, совпадают=true
x=5,0: исходная=5,000000, восстановленная=5,000000, совпадают=true
x=6,0: исходная=6,000000, восстановленная=6,000000, совпадают=true
x=7,0: исходная=7,000000, восстановленная=7,000000, совпадают=true
x=8,0: исходная=8,000000, восстановленная=8,000000, совпадают=true
x=9,0: исходная=9,000000, восстановленная=9,000000, совпадают=true
x=10,0: исходная=10,000000, восстановленная=10,000000, совпадают=true
```